UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/711,148	08/27/2004	Allen Bauer	BOR-008	5147

64107 7590 03/31/2009

KOKKA & BACKUS, PC
200 PAGE MILL ROAD
SUITE 103
PALO ALTO, CA 94306

EXAMINER	
WANG, BEN C	
ART UNIT	PAPER NUMBER
2192	
MAIL DATE	DELIVERY MODE
03/31/2009	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/711,148	BAUER ET AL.	
	Examiner	Art Unit	
	BEN C. WANG	2192	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *12 January 2009*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-46 and 48* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-46 and 48* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. A request for continued examination under 37 CFR 1.114, including the fee set forth in 37 CFR 1.17(e), was filed in this application after final rejection. Since this application is eligible for continued examination under 37 CFR 1.114, and the fee set forth in 37 CFR 1.17(e) has been timely paid, the finality of the previous Office action has been withdrawn pursuant to 37 CFR 1.114. Applicant's submission filed on January 12, 2009 has been entered.

2. Applicant's amendment dated January 12, 2009, responding to the Office Action mailed September 10, 2008 provided in the rejection of claims 1-46 and 48, wherein claims 1 and 25 have been amended.

Claims 1-46 and 48 remain pending in the application and which have been fully considered by the examiner.

Applicant's arguments with respect to claims currently amended have been fully considered but are moot in view of the new grounds of rejection – see *Swart* - art of record, as applied hereto.

Claim Rejections – 35 USC § 103(a)

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

3. Claims 1-46 and 48 are rejected under 35 U.S.C. 103(a) as being unpatentable over Jazdzewski in view of Lam et al., (*.NET® Framework Essentials, June 2001, O'Reilly®*) (hereinafter 'Lam') and Bob Swart, (*Migrating Borland® Delphi™ applications to the Microsoft® .NET Framework with Delphi 8, February 2004, Borland Software Corporation*) (hereinafter 'Swart' - art of record)

4. **As to claim 1** (Currently Amended), Jazdzewski discloses in a form-based development system (e.g., Col. 1, Lines 15-19 - ... a visual development system and methods for improved form-based development of software program), a method for dynamically constructing a form under an object framework during development of an application by a user, the method comprising:

- providing an ancestor class under an object framework, the ancestor class for representing a form in the development system (e.g., Col. 3, Lines 12-26 - ... the user can derive forms from other "ancestor" forms ... Ancestor forms can be any forms already contained in an existing project ...);
- in response to user input, creating a descendant class inheriting from the ancestor class for representing a particular form to be included in the application without directly manipulating metadata of the ancestor form (e.g., Col. 3, Lines 12-26 - ... From inheritance allows the user to create a library of standard form template ... Any changes made to the ancestor form immediately appear in the descendant forms);

- creating a type delegator for the descendant class, thereby enabling the descendant class to track changes made to the particular form during development of the application (e.g., Col. 16, Lines 36-44 – *UpdateChildren* procedure or method serves to propagate an update call to all children … Updating, is an internal housekeeping method which is invoked on a component which is about to be updated. This call adds a component to the update list and invokes the components own updating method; Col. 21, Lines 54-67 - … *AddChild* is a housekeeping method used during creation for adding children … are internal housekeeping methods …);
- creating an instance of the descendant class (e.g., Col. 3, Lines 41-67 – When a form inherits from another form, the system creates a reference to the ancestor from and only generates additional code for adding components and event handlers … The descendant can be modified with no effect on the ancestor …) ;
- tracking changes to the particular form made during development of the application using the type delegator (e.g., Col. 15, Line 23 through Col. 16, Line 28 – Update Manager - … is used for internal housekeeping … *FUpdateList* indicates those components which are being updated; accordingly, it is employed for internal housekeeping during updating operation …);

- persisting information regarding the particular form (e.g., Col. 8, Lines 6-10 – This method causes the data for the "property" to be written out to the persistent image ...);
- subsequently, generating a version of the particular form at runtime based on the persisted information (e.g., Col. 7, Lines 56-62 - ... what happens at design time should mimic exactly what happens at runtime ...)

Further, Jazdzewski discloses a visual development system which allows a user to derive forms from other "ancestor" forms, inheriting their components, properties, and code as a starting point for one's own form (e.g., Abstract) but does not explicitly disclose other limitations stated below.

However, in an analogous art of *.NET Framework® Essentials*, Lam discloses:

- generating intermediate language instructions for creating methods of the descendant class under the object framework (e.g., Overview - ... Web Forms, Windows Forms ...; Sec. 2.5 – Intermediate Language (IL) - 3rd Para); and
- constructing the particular form in the development system based on the instance of the descendant class and making a design time representation of the form visible to the user without using source code and without compiling the descendant class (e.g., Sec. 8.3.3 – Visual Inheritance, 2nd Para – with the advent of Microsoft .NET®, where everything is now object oriented, you can create derived classes by inheriting any base class; since a form in Windows® Forms application

is nothing more than derived class of the base Form class, you can actually derive from your form calls to create other for classes; Sec. 2.4.1 – Assemblies Versus Components – In .NET®, Microsoft® has addressed this confusion by introducing a new concept, assembly, which is a software component that supports plug-and-play, much like a hardware component; Sec. 2.3.1 - Type Libraries on Steroids, 3rd Para – Like the CLR, any application, tool, or utility that can read metadata from a .NET assembly can make of that assembly ...)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Lam into the Jazdzewski's system to further provide other limitations stated above in the Jazdzewski system.

The motivation is that it would further enhance the Jazdzewski's system by taking, advancing and/or incorporating the Lam's system which offers significant advantages to support language integration is such a way that programs can be written in any language, yet can interoperate with one another, taking full advantage of inheritance, polymorphism exceptions, and other features as once suggested by Lam (e.g., Sec. 2.6.2 – The Common Language Specification (CLS), 1st Para)

Furthermore, Lam discloses "Intermediate Language" Code (IL Code) (e.g., Sec. 2.4.3. IL Code) and Intermediate Language (IL) (e.g., Sec. 2.5. Intermediate Language (IL)) but Jazdzewski and Lam do not explicitly disclose other limitations stated below.

However, in an analogous art of *Migrating Borland® Delphi™ applications to the Microsoft® .NET Framework with Delphi 8*, Swart discloses configured to allow migration of one or more tools and one or more programs from another form-based development system to the development system (e.g., Sec. of 'VCL, VCL for .NET, and Windows® Forms', 3rd Para - ... Delphi™ Win 32 projects can be migrated to Delphi™ for .NET with considerable ease ...; Sec. of 'Summary', 1st Para – .... For new .NET applications, you can choose between VCL for .NET and Windows® Forms ...)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Swart into the Jazdzewski-Lam's system to further provide other limitations stated above in the Jazdzewski-Lam system.

The motivation is that it would further enhance the Jazdzewski-Lam's system by taking, advancing and/or incorporating the Swart's system which offers significant advantages that With the release of Delphi™ 8 for the Microsoft® .NET framework, Borland® has enabled Delphi™ developers to target another new platform, supporting the needs of its developer base; Delphi™ for .NET enables developers to write native .NET applications using Windows™ Forms or Web Forms as the framework, or using VCL for .NET components as once suggested by Swart (e.g., Sec. of 'Introduction')

5. **As to claim 2** (Previously Presented) (incorporating the rejection in claim 1), Lam discloses the method wherein the object framework comprises an

infrastructure for building, deploying and running applications having a common language runtime (Sec. 8.3.3 – Visual Inheritance, 2nd Para – with the advent of Microsoft .NET®, where everything is now object oriented, you can create derived classes by inheriting any base class; since a form in Windows® Forms application is nothing more than derived class of the base Form class, you can actually derive from your form calls to create other for classes)

6. **As to claim 3** (Original) (incorporating the rejection in claim 1), Jazdzewski discloses the method wherein said creating step includes creating a descendant class for representing the particular form in a user interface of the development system (e.g., Col. 3, Lines 12-26 - ... the user can derive forms from other "ancestor" forms ... Ancestor forms can be any forms already contained in an existing project ...)

7. **As to claim 4** (Original) (incorporating the rejection in claim 1), Jazdzewski discloses the method wherein said creating step includes inheriting a set of components provided by the ancestor class for representing components that may placed on the particular form (e.g., Col. 3, Lines 12-26 - ... the user can derive forms from other "ancestor" forms ... Ancestor forms can be any forms already contained in an existing project ...)

8. **As to claim 5** (Original) (incorporating the rejection in claim 5), Lam discloses the method wherein said creating step includes creating an assembly

for the descendant class (e.g., Sec. 2.4.1 – Assemblies Versus Components – In .NET®, Microsoft® has addressed this confusion by introducing a new concept, assembly, which is a software component that supports plug-and-play, much like a hardware component)

9. **As to claim 6** (Original) (incorporating the rejection in claim 1), Jazdzewski discloses the method further comprising: creating a second descendant class which inherits from the descendant class, the created second descendant class for representing a form which inherits from the particular form (e.g., Col. 3, Lines 12-26 - ... From inheritance allows the user to create a library of standard form template ... Any changes made to the ancestor form immediately appear in the descendant forms)

10. **As to claim 7** (Original) (incorporating the rejection in claim 1), Jazdzewski discloses the method wherein said constructing step includes constructing the particular form based upon the descendant class in a user interface of the development system (e.g., Fig. 1B, element 200 – Visual Development System; Col. 2, Lines 7-23 - ... Such environments are characterized by an integrated development environment (IDE) providing a form painter, a property getter/setter manager ... a tool palette (with objects which the user can drag and drop on forms) ...; Abstract, Lines 11-13 – Any changes made to the ancestor form immediately appear in the descendant forms)

11. **As to claim 8** (Original) (incorporating the rejection in claim 1), Jazdzewski discloses the method wherein said constructing step includes displaying a component palette including components which the user can select for placement on the particular form (e.g., Figs. 5A-5C)

12. **As to claim 9** (Original) (incorporating the rejection in claim 8), Jazdzewski discloses the method further comprising: receiving user input for placing components selected from the palette on the particular form (e.g., Fig. 3, elements "392" – Form Object, "393" - Properties, and "394" – Events; Col. 7, Lines 28-37 – the inspector 391 comprises an object selector field 392, a properties page 393, and an events page 394)

13. **As to claim 10** (Original) (incorporating the rejection in claim 9), Jazdzewski discloses the method wherein the type delegator tracks creation of components on the particular form in response to a user placing a component on the particular form (e.g., Col. 15, Line 23 through Col. 16, Line 28 – Update Manager - … is used for internal housekeeping … *FUpdateList* indicates those components which are being updated; accordingly, it is employed for internal housekeeping during updating operation …)

14. **As to claim 11** (Original) (incorporating the rejection in claim 9), Jazdzewski discloses the method wherein the type delegator persists information regarding components placed on the particular form, thereby enabling the

components placed on the particular form to be recreated at runtime (e.g., Col. 7, Lines 56-62 - … what happens at design time should mimic exactly what happens at runtime …)

15. **As to claim 12** (Original) (incorporating the rejection in claim 1), Jazdzewski discloses the method wherein said generating step includes generating a constructor for the descendant class (e.g., Col. 21, Lines 54-55 – After declaring a constructor (Create) ...)

16. **As to claim 13** (Original) (incorporating the rejection in claim 12), Lam discloses the method wherein said step of generating a constructor includes generating intermediate language instructions for building the constructor (e.g., Sec. 2.5 – Intermediate Language (IL), 3rd Para – Microsoft® calls its own language-abstraction layer the Common Intermediate Language (CIL); Similar bytecode, IL supports all object-oriented features, including data abstraction, inheritance, polymorphism and useful concepts such as exceptions and events; any .NET® language may be converted into IL, so .NET® supports multiple languages and perhaps multiple platforms in the future [as long as the target platforms have a CLR])

17. **As to claim 14** (Original) (incorporating the rejection in claim 13), Lam discloses the method wherein said step of generating intermediate language instructions includes using classes provided by the object framework for

generating intermediate language instructions constructor (e.g., Sec. 2.5 – Intermediate Language (IL), 3rd Para – Microsoft® calls its own language-abstraction layer the Common Intermediate Language (CIL); Similar bytecode, IL supports all object-oriented features, including data abstraction, inheritance, polymorphism and useful concepts such as exceptions and events; any .NET® language may be converted into IL, so .NET® supports multiple languages and perhaps multiple platforms in the future [as long as the target platforms have a CLR])

18. **As to claim 15** (Original) (incorporating the rejection in claim 13), Lam discloses the method wherein said generating step includes generating instructions for calling the constructor of the ancestor class, thereby ensuring execution of an appropriate constructor implemented by the ancestor class (e.g., Sec. 2.5 – Intermediate Language (IL), 3rd Para – Microsoft® calls its own language-abstraction layer the Common Intermediate Language (CIL); Similar bytecode, IL supports all object-oriented features, including data abstraction, inheritance, polymorphism and useful concepts such as exceptions and events; any .NET® language may be converted into IL, so .NET® supports multiple languages and perhaps multiple platforms in the future [as long as the target platforms have a CLR])

19. **As to claim 16** (Original) (incorporating the rejection in claim 1), Lam discloses the method wherein said generating step includes generating methods

for overriding notification methods of the ancestor class (e.g., Sec. 8.2.2.1 – Extending existing controls – because Windows® Forms API is object oriented, extending controls is as easy as deriving from the default behavior of the control)

20. **As to claim 17** (Original) (incorporating the rejection in claim 16), Lam discloses the method wherein said generating step includes generating intermediate language instructions for overriding notification methods of the ancestor class (e.g., Sec. 8.2.2.1 – Extending existing controls – because Windows® Forms API is object oriented, extending controls is as easy as deriving from the default behavior of the control)

21. **As to claim 18** (Original) (incorporating the rejection in claim 1), Jazdzewski discloses the method wherein the type delegator provides information for enumerating fields, methods, properties, and events in response to user input on the particular form in the development system (e.g., Col. 16, Lines 36-44 – *UpdateChildren* procedure or method serves to propagate an update call to all children … Updating, is an internal housekeeping method which is invoked on a component which is about to be updated. This call adds a component to the update list and invokes the components own updating method; Col. 21, Lines 54-67 - … *AddChild* is a housekeeping method used during creation for adding children … are internal housekeeping methods …)

22. **As to claim 19** (Original) (incorporating the rejection in claim 1), Lam discloses the method wherein the type delegator generates metadata information in response to user input on the particular form (e.g., Sec. 2.3 – Metadata, 1st Para. – Metadata is machine-readable information about a resource, or "data about data"; Such information might include details on content, format, size, or other characteristics of a data source; In .NET®, metadata includes type definitions, version information, external assembly references, and other standardized information; Sec. 2.3.1, 1st Para. – Just as type libraries are C++ header files on steroids, metadata is a type library on steroids; in .NET® metadata is a common mechanism or dialect that the .NET® runtime, compilers, and tools can all use. Microsoft .NET® uses metadata to describe all types that are used and exposed by a particular .NET® assembly; much richer than a type library, metadata includes descriptions of an assembly and modules, classes, interfaces, methods, properties fields, events, global methods, and so forth)

23. **As to claim 20** (Original) (incorporating the rejection in claim 19), Lam discloses the method wherein said step of generating metadata information includes adding a reference to methods of the application assigned to components on the particular form (e.g., Sec. 2.3 – Metadata, 1st Para. – Metadata is machine-readable information about a resource, or "data about data"; Such information might include details on content, format, size, or other characteristics of a data source; In .NET®, metadata includes type definitions, version information, external assembly references, and other standardized

information; Sec. 2.3.1, 1st Para. – Just as type libraries are C++ header files on steroids, metadata is a type library on steroids; in .NET® metadata is a common mechanism or dialect that the .NET® runtime, compilers, and tools can all use. Microsoft .NET® uses metadata to describe all types that are used and exposed by a particular .NET® assembly; much richer than a type library, metadata includes descriptions of an assembly and modules, classes, interfaces, methods, properties fields, events, global methods, and so forth)

24. **As to claim 21** (Original) (incorporating the rejection in claim 1), Jazdzewski discloses the method further comprising: persisting state of the particular form, enabling the particular form to be recreated at runtime (e.g., Col. 7, Lines 56-62 - … what happens at design time should mimic exactly what happens at runtime …)

25. **As to claim 22** (Original) (incorporating the rejection in claim 21), Jazdzewski discloses the method wherein said persisting step includes persisting user input on the particular form (e.g., Col. 8, Lines 6-10 – This method causes the data for the "property" to be written out to the persistent image …)

26. **As to claim 23** (Original) (incorporating the rejection in claim 1), please refer to claim **1** above, accordingly.

27. **As to claim 24** (Original) (incorporating the rejection in claim 1), Jazdzewski discloses a downloadable set of processor-executable instructions for performing the method (e.g., Abstract - A visual development system which allows a user to derive forms from other "ancestor" forms, inheriting their components, properties, and code as a starting point for one's own forms ...)

28. **As to claim 25** (Currently Amended), Jazdzewski discloses a development system for dynamically constructing a form responsive to user input under an object framework during development of an application (e.g., Col. 1, Lines 15-19 - ... a visual development system and methods for improved form-based development of software program), the system comprising:

- a computer having a processor and memory;
- an ancestor class for representing the form under the object framework (e.g., Col. 3, Lines 12-26 - ... the user can derive forms from other "ancestor" forms ... Ancestor forms can be any forms already contained in an existing project ...);
- a proxy module for creating a descendant class inheriting from the ancestor class in response to user input, dynamically generating methods of the descendant class, and constructing an instance of the descendant class under the object framework for representing the form in the development system (e.g., Col. 3, Lines 12-26 - ... From inheritance allows the user to create a library of standard form template ... Any

changes made to the ancestor form immediately appear in the descendant forms);

- a type delegator for the descendant class for tracking user input on the form during development of the application (e.g., Col. 16, Lines 36-44 – *UpdateChildren* procedure or method serves to propagate an update call to all children ... Updating, is an internal housekeeping method which is invoked on a component which is about to be updated. This call adds a component to the update list and invokes the components own updating method; Col. 21, Lines 54-67 - ... *AddChild* is a housekeeping method used during creation for adding children ... are internal housekeeping methods ...);
- a persistence mechanism for persisting user input on the form (e.g., Col. 8, Lines 6-10 – This method causes the data for the "property" to be written out to the persistent image ...);
- a module for displaying a design time representation of the form in a user interface of the development system based on the descendant class (e.g., Col. 3, Lines 41-67 – When a form inherits from another form, the system creates a reference to the ancestor from and only generates additional code for adding components and event handlers ... The descendant can be modified with no effect on the ancestor ...) and the persisted user input (e.g., Col. 8, Lines 6-10 – This method causes the data for the "property" to be written out to the persistent image ...) and subsequently generating a version of the form at runtime based on the persisted information (e.g.,

Col. 7, Lines 56-62 - … what happens at design time should mimic exactly what happens at runtime …)

Further, Jazdzewski discloses a visual development system which allows a user to derive forms from other "ancestor" forms, inheriting their components, properties, and code as a starting point for one's own form (e.g., Abstract) but does not explicitly disclose other limitations stated below.

However, in an analogous art of *.NET Framework® Essentials*, Lam discloses:

- without using source code and without compiling the descendant class (e.g., Sec. 2.4.1 – Assemblies Versus Components – In .NET®, Microsoft® has addressed this confusion by introducing a new concept, assembly, which is a software component that supports plug-and-play, much like a hardware component; Sec. 2.3.1 - Type Libraries on Steroids, 3rd Para – Like the CLR, any application, tool, or utility that can read metadata from a .NET assembly can make of that assembly …)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Lam into the Jazdzewski's system to further provide other limitations stated above in the Jazdzewski system.

The motivation is that it would further enhance the Jazdzewski's system by taking, advancing and/or incorporating the Lam's system which offers significant advantages to support language integration is such a way that programs can be

written in any language, yet can interoperate with one another, taking full advantage of inheritance, polymorphism exceptions, and other features as once suggested by Lam (e.g., Sec. 2.6.2 – The Common Language Specification (CLS), 1st Para)

Furthermore, Lam discloses "Intermediate Language" Code (IL Code) (e.g., Sec. 2.4.3. IL Code) and Intermediate Language (IL) (e.g., Sec. 2.5. Intermediate Language (IL)) but Jazdzewski and Lam do not explicitly disclose other limitations stated below.

However, in an analogous art of *Migrating Borland® Delphi™ applications to the Microsoft® .NET Framework with Delphi 8*, Swart discloses:

- configured to allow migration of one or more tools and one or more programs from another form-based development system to the development system (e.g., Sec. of 'VCL, VCL for .NET, and Windows® Forms', 3rd Para - ... Delphi™ Win 32 projects can be migrated to Delphi™ for .NET with considerable ease ...; Sec. of 'Summary', 1st Para – .... For new .NET applications, you can choose between VCL for .NET and Windows® Forms ...)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Swart into the Jazdzewski-Lam's system to further provide other limitations stated above in the Jazdzewski-Lam system.

The motivation is that it would further enhance the Jazdzewski-Lam's system by taking, advancing and/or incorporating the Swart's system which offers

significant advantages that With the release of Delphi™ 8 for the Microsoft® .NET framework, Borland® has enabled Delphi™ developers to target another new platform, supporting the needs of its developer base; Delphi™ for .NET enables developers to write native .NET applications using Windows™ Forms or Web Forms as the framework, or using VCL for .NET components as once suggested by Swart (e.g., Sec. of 'Introduction')

29. **As to claim 26** (Previously Presented) (incorporating the rejection in claim 25), please refer to claim **2** above.

30. **As to claim 27** (Original) (incorporating the rejection in claim 25), please refer to claim **5** above.

31. **As to claim 28** (Previously Presented) (incorporating the rejection in claim 25), please refer to claim **3** above.

32. **As to claim 29** (Previously Presented) (incorporating the rejection in claim 28), Jazdzewski discloses the system wherein the descendant class inherits a set of components provided by the ancestor class for representing components that may be placed on the form (e.g., Abstract, Lines 1-4 – a visual development system is described which allows a user to derive forms from other "ancestor" forms, inheriting their components, properties, and code as a starting point for one's own forms)

33. **As to claim 30** (Original) (incorporating the rejection in claim 28), please refer to claim **6** above.

34. **As to claim 31** (Original) (incorporating the rejection in claim 28), Jazdzewski discloses the system wherein the form includes a component palette comprising components which the user can select (e.g., Fig. 3; Col. 6, Lines 36-37 – Fig. 3 illustrates an application development environment, which is provided by Delphi®; Lines 41-46 – as show, the programming environment 360 comprises an main window 361, a form 371, a code editor window 381, and an object manager or "inspector" window 391; the main window 361 itself comprises main menu 361, tool bar buttons 363, and component palette 364; main menu 362 lists user-selectable commands, in a conventional manner)

35. **As to claim 32** (Original) (incorporating the rejection in claim 31), please refer to claim **22** above.

36. **As to claim 33** (Original) (incorporating the rejection in claim 32), please refer to claim **21** above.

37. **As to claim 34** (Original) (incorporating the rejection in claim 25), please refer to claim **12** above.

38. **As to claim 35** (Original) (incorporating the rejection in claim 34), please refer to claim **13** above.

39. **As to claim 36** (Original) (incorporating the rejection in claim 35), please refer to claim **14** above.

40. **As to claim 37** (Original) (incorporating the rejection in claim 35), please refer to claim **15** above.

41. **As to claim 38** (Original) (incorporating the rejection in claim 25), please refer to claim **16** above.

42. **As to claim 39** (Original) (incorporating the rejection in claim 38), please refer to claim **17** above.

43. **As to claim 40** (Original) (incorporating the rejection in claim 39), Lam discloses the system wherein the proxy module generates intermediate language instructions using classes for generating intermediate language instructions provided by the object framework (e.g., Sec. 2.5 – Intermediate Language (IL), 3rd Para – Microsoft® calls its own language-abstraction layer the Common Intermediate Language (CIL); Similar bytecode, IL supports all object-oriented features, including data abstraction, inheritance, polymorphism and useful concepts such as exceptions and events; any .NET® language may be

converted into IL, so .NET® supports multiple languages and perhaps multiple platforms in the future [as long as the target platforms have a CLR])

44. **As to claim 41** (Original) (incorporating the rejection in claim 25), please refer to claim **18** above.

45. **As to claim 42** (Original) (incorporating the rejection in claim 25), please refer to claim **19** above.

46. **As to claim 43** (Original) (incorporating the rejection in claim 42), please refer to claim **20** above.

47. **As to claim 44** (Original) (incorporating the rejection in claim 43), Lam discloses the system wherein said metadata information and the descendant class are used to reconstruct the form as part of the application at runtime (e.g., Sec. 2.3 – Metadata, 1st Para. – Metadata is machine-readable information about a resource, or "data about data"; Such information might include details on content, format, size, or other characteristics of a data source; In .NET®, metadata includes type definitions, version information, external assembly references, and other standardized information; Sec. 2.3.1, 1st Para. – Just as type libraries are C++ header files on steroids, metadata is a type library on steroids; in .NET® metadata is a common mechanism or dialect that the .NET® runtime, compilers, and tools can all use. Microsoft .NET® uses metadata to

describe all types that are used and exposed by a particular .NET® assembly; much richer than a type library, metadata includes descriptions of an assembly and modules, classes, interfaces, methods, properties fields, events, global methods, and so forth)

48. **As to claim 45** (Original) (incorporating the rejection in claim 25), Jazdzewski discloses the system wherein the form comprises a form open on a visual design surface of the development system (e.g., Fig. 1B, element 200 – Visual Development System; Col. 2, Lines 7-23 - ... Such environments are characterized by an integrated development environment (IDE) providing a form painter, a property getter/setter manager ... a tool palette (with objects which the user can drag and drop on forms) ...; Abstract, Lines 11-13 – Any changes made to the ancestor form immediately appear in the descendant forms)

49. **As to claim 46** (Previously Presented) (incorporating the rejection in claim 25), Jazdzewski discloses the system, wherein the persisting persists state of the form (e.g., Col. 8, Lines 6-10 – This method causes the data for the "property" to be written out to the persistent image ...)

50. **As to claim 48** (Original) (incorporating the rejection in claim 46), Jazdzewski discloses the system wherein the persisting mechanism enables the form to be recreated at runtime as part of the application (e.g., Col. 7, Lines 56-

62 - ... what happens at design time should mimic exactly what happens at runtime ...)

Conclusion

51. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Ben C. Wang whose telephone number is 571-270-1240. The examiner can normally be reached on Monday - Friday, 8:00 a.m. - 5:00 p.m., EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on 571-272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/Ben C Wang/
Ben C. Wang
Examiner, Art Unit 2192

/Tuan Q. Dam/
Supervisory Patent Examiner, Art Unit 2192